МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ПОЛІЩУК ДМИТРО ОЛЕКСАНДРОВИЧ

Допускається до захисту:

в.о. завідувача кафедри
інформаційних технологій

канд. техн. наук, доцент

_____ О. В. Зелінська

«____»_____ 2025 р.

ВЕБДОДАТОК ДЛЯ УПРАВЛІННЯ ДІЯЛЬНІСТЮ АВТОСАЛОНУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Р. М. Бабаков, професор кафедри
інформаційних технологій,
к. т. н., доцент

_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Поліщук Д. О. Вебдодаток для управління діяльністю автосалону. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано сучасні онлайн-платформи продажу авто, та створено конкурентоспроможний веб-сайт автосалону і додаток для керування ним.

ABSTRACT

Polishchuk D. O. Web application for managing the activities of a car dealership. Specialty 122 "Computer Science", educational program "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia 2025.

In the qualification (bachelor's) thesis, modern online car sales platforms were researched and analyzed, and a competitive car dealership website and an application for managing it were created.

ЗМІСТ

ВСТУП

У сучасному діловому середовищі цифровізація процесів стала невід'ємною частиною успішного функціонування підприємств. Автомобільна індустрія, зокрема діяльність автосалонів, не є винятком. В умовах зростаючої конкуренції, високих вимог клієнтів та стрімкого розвитку інформаційних технологій, ефективне управління ресурсами автосалону вимагає впровадження сучасних програмних рішень. Одним із таких рішень є вебдодатки – потужний інструмент, що дозволяє автоматизувати облік, оптимізувати бізнес-процеси, покращити комунікацію з клієнтами та підвищити загальний рівень обслуговування.

Традиційні методи управління діяльністю автосалону, які базуються на паперовому документообігу або застарілих програмних засобах, втрачають свою актуальність через обмежену функціональність, складність масштабування та низький рівень інтеграції з іншими системами. Вебдодатки, на відміну від настільного програмного забезпечення, мають переваги у вигляді доступності з будь-якого пристрою з доступом до Інтернету, зручного інтерфейсу користувача, можливості хмарного зберігання даних та інтеграції з зовнішніми сервісами, такими як платіжні системи, сервіси доставки або CRM-платформи.

Метою цієї роботи є розробка вебдодатку для управління діяльністю автосалону, який забезпечуватиме зручне керування товарними запасами, клієнтською базою, продажами та аналітикою. Основна увага приділяється створенню інтуїтивно зрозумілого та адаптивного інтерфейсу, побудові надійної архітектури системи та забезпеченню високого рівня безпеки даних.

Реалізація такого додатку дозволить не лише оптимізувати внутрішні процеси автосалону, а й покращити взаємодію з клієнтами, підвищити простоту роботи персоналу та сприяти загальному зростанню прибутковості бізнесу.

РОЗДІЛ 1 ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ВЕБДОДАТКУ ДЛЯ УПРАВЛІННЯ ДІЯЛЬНІСТЮ АВТОСАЛОНУ

1.1.Аналіз поточних проблем у діяльності автосалонів без автоматизованих систем

Діяльність сучасного автосалону охоплює великий спектр завдань: реалізація автомобілів, обслуговування клієнтів, управління складом, фінансовий облік, аналітика ринку, маркетингова діяльність, взаємодія з постачальниками, ведення гарантійного обслуговування тощо. У багатьох випадках, особливо в середніх та малих компаніях, ці процеси реалізуються із застосуванням ручного або фрагментарного підходу, часто без єдиної інформаційної системи. Такий підхід має низку суттєвих недоліків.

Однією з найсуттєвіших проблем є ручне введення інформації та використання окремих таблиць Excel, які не мають захисту від дублювання чи втрати даних. Виникає ситуація, коли менеджери мають різні версії одного й того самого документа, або дані змінюються одночасно без синхронізації. Такі помилки можуть призвести до:

- подвоєння замовлень;
- втрати клієнтських заявок;
- помилок у наявності автомобілів на складі;
- фінансових неточностей.

Це все знижує якість обслуговування, створює конфлікти з клієнтами та може завдати репутаційної шкоди.

Співробітники автосалону витрачають надто багато часу на ручну роботу:

- заповнення форм;
- пошук потрібної інформації в документах;
- створення звітів;

- погодження між відділами без централізованої системи.

Усе це зменшує ефективність виконання основних функцій працівників і сповільнює процес обслуговування клієнтів. Менеджери не мають інструментів для швидкого аналізу ситуації та прийняття оперативних рішень.[1,4,5,8]

Відділ продажу, технічне обслуговування, бухгалтерія, склад, маркетинг часто використовують різні програми або навіть ручні методи. В результаті відсутня цілісна картина діяльності компанії. Інформація дублюється або втрачається. Це створює:

- затримки у погодженні дій;
- складність у відстеженні життєвого циклу клієнта;
- конфлікти через недоступність або невчасне оновлення інформації.

Автосалон без автоматизованої системи не має можливості повноцінно аналізувати свою діяльність.[9] Брак аналітичних інструментів унеможливлює виявлення неефективних ділянок у роботі, таких як:

- непродуктивні співробітники;
- моделі авто, які не продаються;
- перевитрати бюджету;
- зниження попиту.

Таблиця 1.1 Порівняння наслідків неавтоматизованих і автоматизованих процесів

Категорія	Без автоматизації	З автоматизацією
Введення даних	Ручне, схильне до помилок	Автоматизоване, перевірки, логіка
Робочий час персоналу	Неоптимальне використання	Звільнення час для важливих задач
Взаємодія відділів	Нескоординована	Централізована, прозора
Контроль та аналітика	Відмінний або примітивний	Повноцінні звіти, KPI, прогнози

[4,7]

1.2.Переваги використання вебдодатку у порівнянні з традиційними методами та іншими типами ПЗ

Використання вебдодатку для управління автосалоном відкриває нові можливості для оптимізації внутрішніх процесів, підвищення ефективності працівників і якості обслуговування клієнтів.[6] Основні переваги такого підходу розглянемо нижче.

Вебдодаток забезпечує цілодобовий доступ до системи з будь-якої точки світу, де є інтернет-з'єднання. Це особливо важливо для менеджерів, які повинні оперативно приймати рішення, або власників, що здійснюють моніторинг бізнесу віддалено.

Вебтехнології також дозволяють легко масштабувати систему відповідно до розширення бізнесу: додавання нових користувачів, філій, розділів чи функцій не вимагає капітального втручання в структуру додатку.

Автоматизація всіх етапів взаємодії з клієнтом — від первинного звернення до післяпродажного обслуговування — дає змогу створити єдину

базу даних, в якій інформація постійно оновлюється та є доступною для всіх учасників процесу.

- Менеджер з продажу бачить історію спілкування з клієнтом.
- Складський працівник отримує інформацію про резервування товару.
- Бухгалтер має доступ до фінансових операцій у режимі реального часу.

Однією з головних переваг вебдодатку є можливість створення інтерактивних звітів, графіків і панелей моніторингу (dashboard). Завдяки цьому можна:

- відстежувати ефективність продажів;
- аналізувати динаміку клієнтського попиту;
- виявляти слабкі місця в маркетингових кампаніях;
- прогнозувати попит і формувати складські замовлення на основі аналітики.

Таблиця 1.2 Порівняльна ефективність традиційного управління та вебдодатку

Критерій	Традиційний підхід	Вебдодаток
Час на обробку заявки	До 2 годин	Менше 5 хвилин
Можливість аналізу даних	Обмежена, ручна	Автоматизовані звіти
Доступність з різних пристроїв	Обмежена	Повна
Інтеграція з CRM, ERP	Вимагає додаткового ПЗ	Повна, вбудована
Зворотній зв’язок від клієнтів	Через телефон, пошту	Онлайн-форми, чат, автоаналіз заявок

Автосалони, що використовують сучасні IT-рішення, здатні швидше реагувати на зміни ринку, краще адаптуватися до потреб клієнтів і

забезпечувати вищу якість сервісу. Це формує довіру клієнтів і зміцнює позиції на ринку.[2,3,4,10]

1.3. Очікуваний вплив впровадження вебдодатку на ефективність бізнес-процесів автосалону

Впровадження сучасного вебдодатку для управління автосалоном має на меті не лише автоматизацію рутинних завдань, а й значне підвищення ефективності усіх бізнес-процесів. У цьому розділі розглянемо вплив автоматизованої системи на ключові аспекти діяльності автосалону.

Автоматизація процесів дозволяє значно зменшити витрати часу на виконання рутинних дій. Це означає, що співробітники можуть зосередитися на більш стратегічних і креативних завданнях, зокрема на підвищенні рівня обслуговування клієнтів, розвитку партнерств і вдосконаленні бізнес-процесів. Крім того, система автоматично розподіляє задачі між співробітниками, відстежує прогрес і забезпечує прозорість виконання. Це дозволяє керівникам оперативно виявляти слабкі місця та вживати відповідних заходів.[1,2]

У конкурентному середовищі ринку автомобілів рівень клієнтського сервісу є вирішальним чинником успіху. Вебдодаток забезпечує інтеграцію з клієнтськими базами, що дозволяє персоналізувати підхід до кожного клієнта. Система автоматично зберігає історію взаємодії з клієнтами, їхні переваги та запити, що дозволяє:

- Пропонувати індивідуальні знижки та бонуси;
- Надсилати релевантні маркетингові повідомлення;
- Здійснювати проактивний супровід клієнта після продажу.

Це сприяє формуванню довгострокових відносин з клієнтами та повторним продажам.

Оскільки ручне ведення документів пов'язане з високим ризиком помилок, вебдодаток мінімізує ці ризики завдяки автоматичним перевіркам та валідації даних. Впровадження єдиної бази даних забезпечує консистентність і цілісність інформації, а централізоване збереження документів – зручність доступу та захист від втрати.

Система також передбачає механізми резервного копіювання, що дозволяє зберегти дані навіть у разі технічних збоїв або помилок користувачів.

Автоматизована система обліку дозволяє в режимі реального часу відстежувати всі переміщення транспортних засобів, стан замовлень та наявність авто. Це дає можливість:

- Аналізувати попит та формувати оптимальні запаси;
- Автоматично генерувати заявки на постачання;
- Скоротити втрати, пов’язані з простоєм автомобілів на складі.

Таким чином, автосалон отримує контроль над логістикою та здатність швидко реагувати на зміни ринку.[4]

Контроль за фінансовими потоками – одна з головних переваг впровадження вебдодатку. Він дозволяє вести детальний облік доходів і витрат, відображати всі операції в режимі реального часу, а також формувати звіти за будь-який період.

Система дозволяє аналізувати фінансову ефективність роботи як у цілому автосалону, так і окремих його підрозділів. Це підвищує прозорість фінансів і знижує ймовірність шахрайства або нецільового використання коштів.

Автоматизовані системи дозволяють збирати та аналізувати велику кількість даних, які стають базою для прийняття ефективних управлінських рішень. Керівництво автосалону може відстежувати:

- Динаміку продажів;
- Показники ефективності кожного співробітника;
- Рентабельність певних видів послуг або моделей авто.

На основі цієї інформації розробляються стратегії розвитку, плануються бюджети, розподіляються ресурси та коригується маркетингова політика.

Вебдодаток дозволяє централізувати роботу всієї системи в хмарному середовищі, що зменшує потребу в локальному IT-обладнанні, витратах на обслуговування серверів та оновлення програмного забезпечення. Це забезпечує гнучкість масштабування системи та мінімізацію витрат на її підтримку.[6,8]

Автосалони, які впроваджують сучасні IT-рішення, мають перевагу на ринку:

- Краща взаємодія з клієнтами;
- Вища швидкість обслуговування;
- Прозорість і контроль усіх процесів.

Ці фактори дозволяють компанії не лише зберегти позиції, а й активно розширювати ринки, залучати нових клієнтів та партнерів.

- Знизити витрати часу та ресурсів;
- Зменшити помилки та втрати;
- Підвищити задоволення клієнтів;
- Збільшити прибутковість та конкурентоспроможність;
- Отримати прозору аналітику та контроль за процесами;
- Зменшити витрати на IT-інфраструктуру.

Таким чином, автоматизація процесів через спеціалізоване програмне забезпечення є не просто доцільною, а й стратегічно необхідною для сталого розвитку сучасного автосалону.[2,7]

1.4.Роль вебдодатку у формуванні клієнтоорієнтованого підходу в автобізнесі

У сучасних умовах конкуренції головною цінністю для автосалону стає задоволений клієнт. Вебдодаток може стати основою клієнтоорієнтованої стратегії, забезпечуючи персоналізацію обслуговування, прозорість взаємодії та швидкий зворотний зв'язок. Це дозволяє автосалонам не лише залучати нових клієнтів, але й утримувати постійних, формуючи лояльність до бренду.

Інтегровані інструменти для онлайн-чату, повідомлень, автоматичних відповідей на електронну пошту, push-нотифікацій у вебдодатку дозволяють оперативно відповідати на запити клієнтів. Крім того, вебдодаток може містити віртуального помічника (бота), який 24/7 допомагає у виборі автомобіля, записі на тест-драйв або сервіс.

Аналіз даних про попередні покупки, інтереси, активність у додатку дозволяє формувати персоналізовані знижки, рекомендації та нагадування, що підвищують імовірність повторних звернень. Таким чином вебдодаток трансформується з інструменту продажу в інструмент підтримки життєвого циклу клієнта.

Цифрові сервіси дають клієнту можливість самостійно переглядати статус замовлення, вартість послуг, наявність товарів на складі. Такий рівень відкритості підвищує довіру до компанії, знижує кількість конфліктних ситуацій та покращує загальне враження від співпраці.

Замість традиційного візиту до автосалону вебдодаток дозволяє реалізувати повний цикл взаємодії: від вибору авто, перевірки наявності, отримання консультації до замовлення й оплати. Це значно економить час клієнта та робить процес зручним, особливо для молодшого покоління покупців.

Задоволені користувачі вебдодатку мають більшу схильність залишати позитивні відгуки в онлайн-просторі, рекомендувати сервіс знайомим. Таким чином цифровий продукт стимулює природне зростання клієнтської бази.

Формування клієнтоорієнтованого підходу за допомогою вебдодатку — стратегічна перевага сучасного автосалону. Інструменти персоналізації, прозорості, цифрової комунікації та самообслуговування формують позитивний клієнтський досвід, який неможливо досягти лише за допомогою традиційних каналів.

Розробка такого вебдодатку — це не просто технологічне вдосконалення, а нова філософія ведення бізнесу, що відповідає очікуванням споживача цифрової епохи.[1,4,10]

РОЗДІЛ 2 ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ІНСТРУМЕНТІВ

2.1. Постановка задачі

Після глибокого аналізу існуючих проблем у діяльності автосалонів без автоматизованих систем (розділ 1.1), а також враховуючи обґрунтування доцільності розробки вебдодатку (розділ 1.2), постає практичне завдання розробити інноваційне рішення, яке забезпечить як якісну присутність автосалону в цифровому середовищі, так і ефективну організацію внутрішніх процесів.

У зв’язку з цим виникає потреба створити сучасний вебдодаток, який стане інструментом інтеграції бізнес-процесів, аналітики, управління клієнтським досвідом та онлайн-продажів. Проєкт повинен відповідати принципам цифрової трансформації, описаним у попередніх розділах, та бути готовим до масштабування, аналітичного розвитку й адаптації до нових технологічних викликів.

Сучасний автомобільний ринок вимагає від автосалонів не лише якісного продукту, а й високого рівня цифрової присутності. Покупці дедалі частіше шукають авто онлайн, очікують зручного перегляду моделей, прозорих умов покупки та можливості здійснити попереднє замовлення без візиту до салону. Водночас працівники автосалонів потребують ефективного інструменту для обробки заявок, контролю продажів і актуалізації товарного каталогу.

З метою задоволення цих потреб і підвищення ефективності бізнес-процесів постає завдання розробити вебдодаток, який включатиме як публічну частину (сайт для покупців), так і адміністративну панель (інтерфейс для менеджерів та адміністрації).

Публічна частина вебдодатку повинна реалізувати наступний функціонал:

- Зручний та інформативний каталог автомобілів з технічними характеристиками, фото та цінами;
- Онлайн-форма для заповнення заявки на купівлю або тест-драйв автомобіля;
- Адаптивність для всіх типів пристроїв — смартфонів, планшетів, ПК.
- Адміністративна частина повинна містити:
- Таблицю з усіма отриманими заявками, з можливістю сортування та обробки звернень;
- Панель для управління каталогом автомобілів (додавання нових моделей, редагування описів, оновлення наявності);
- Модуль статистики та звітності, який відображає динаміку продажів, кількість оброблених заявок, популярні моделі тощо;
- Безпечну систему входу з розмежуванням прав доступу.

Мета проєкту**:** створити повнофункціональний вебдодаток, який об'єднає ефективний цифровий канал продажу авто та зручний інструмент управління діяльністю автосалону для адміністрації.

Для досягнення цієї мети необхідно реалізувати такі задачі:

1. Провести аналіз потреб цільової аудиторії (покупців авто та менеджерів автосалону);
2. Визначити архітектуру вебдодатку, розробити інтерфейси користувача для обох частин системи;
3. Реалізувати функціонал перегляду моделей, фільтрації та створення заявок у публічному модулі;
4. Розробити адміністративну панель з системою управління контентом і модулем звітності;

5. Забезпечити безпечне зберігання даних, інтеграцію з базою даних і можливість подальшого масштабування;

6. Провести тестування функціоналу, виправити виявлені недоліки, оптимізувати продуктивність системи;

7. Підготувати інструкції для користувачів та адміністрації, впровадити додаток у роботу автосалону.

Проєкт є прикладом цифрової трансформації в сфері продажу автомобілів, що має на меті підвищити зручність, ефективність та прозорість усіх ключових процесів.

2.2. Розробка схеми алгоритму роботи вебдодатку для управління діяльністю автосалону

Основною метою є побудова логічної структури функціонування вебдодатку, яка охоплює всі ключові етапи взаємодії користувача з системою, а також забезпечує ефективну взаємодію між публічною та адміністративною частинами додатку. Алгоритм покликаний відобразити послідовність процесів від перегляду автомобіля до обробки заявки адміністратором, а також дії, пов’язані з оновленням каталогу й аналітикою.

Основні компоненти алгоритму

Для клієнта (користувача сайту):

- Перехід на головну сторінку сайту;
- Ознайомлення з каталогом автомобілів;
- Вибір моделі та перегляд детальної інформації;
- Натискання кнопки «Оформити заявку»;
- Заповнення контактної інформації;
- Надсилання заявки.

Для адміністратора (менеджера автосалону):

- Вхід до адмінпанелі;
- Перегляд списку нових заявок;
- Обробка заявки (зв’язок з клієнтом);
- Додавання нових моделей у каталог (назва, характеристики, фото);
- Перегляд звітів з продажу авто.

Алгоритм умовно поділяється на дві гілки: користувацьку і адміністративну. Вони функціонують паралельно, взаємодіючи через базу даних.

Основні блоки:

- Взаємодія з інтерфейсом користувача (UI);
- Формування та надсилання запиту до серверу;
- Обробка запиту та взаємодія з базою даних;
- Оновлення інтерфейсу адміністратора;
- Збереження даних заявки / оновлення каталогу / формування звіту.

Всі заявки надсилаються у вигляді структурованих об’єктів (наприклад, JSON) на сервер, де обробляються бекенд-логікою. У відповідь адміністратор отримує оновлену інформацію у вигляді дашборду із заявками.

Оновлення каталогу працює за тим же принципом — дані про нову модель додаються до бази даних, після чого автоматично оновлюється вміст публічного сайту.

Для безпеки алгоритму передбачено наступні заходи:

- Захист API-запитів через токенізацію та автентифікацію;
- Вхід до адмінпанелі лише за логіном і паролем;
- Переваги чітко структурованого алгоритму

- Зниження кількості помилок у роботі системи;
- Прискорення відповіді на дії користувача;
- Зручність масштабування й розширення функціоналу;
- Підвищення рівня безпеки даних;
- Можливість тестування окремих блоків алгоритму для контролю стабільності.

Розробка схеми алгоритму роботи вебдодатку є критичним етапом, який забезпечує логічну цілісність функціоналу системи. Це дозволяє як користувачам, так і адміністраторам ефективно й безпечно взаємодіяти з платформою, зменшуючи ризики помилок і підвищуючи якість обслуговування. Алгоритм може слугувати основою для подальшої документації, оптимізації й масштабування цифрового рішення.[3,11]

2.3. Огляд інструментів створення вебдодатку для управління діяльністю автосалону

Для реалізації вебдодатку, що об’єднує публічний інтерфейс користувача та адміністративну панель автосалону, було обрано стек сучасних технологій, який забезпечує гнучкість, масштабованість, зручність розробки та високий рівень безпеки. У цьому розділі розглянуто ключові інструменти, використані при створенні вебдодатку.

Backend: Django

Django — це високорівневий фреймворк на мові Python, який дозволяє швидко розробляти безпечні та масштабовані вебдодатки.

Основні переваги Django:

- Система адміністрування з готовим інтерфейсом для управління контентом;
- ORM (Object-Relational Mapping) для інтеграції з базами даних, зокрема PostgreSQL;
- Вбудовані механізми захисту від SQL-ін’єкцій, CSRF, XSS;
- Підтримка REST API за допомогою Django REST Framework (DRF);
- Велика екосистема плагінів та активна спільнота розробників;
- Швидка побудова CRUD-операцій та інтеграція з зовнішніми сервісами.

Django був використаний для обробки заявок, створення API, управління користувачами, реєстрації дій у системі та забезпечення доступу до адмінпанелі. Завдяки своїй модульній структурі він дозволяє масштабувати систему за рахунок додавання нових функціональностей без переписування наявного коду.[12-15]

Frontend: React + TypeScript + Tailwind CSS

React — бібліотека JavaScript для створення користувацьких інтерфейсів, що дозволяє реалізовувати динамічні та інтерактивні компоненти. У поєднанні з TypeScript, який додає суворі типи до JavaScript, забезпечується висока стабільність коду та полегшення командної розробки.

Основні переваги:

- Компонентна структура дозволяє розділити інтерфейс на незалежні блоки, які легко тестувати та перевикористовувати;
- Зворотній зв'язок у режимі реального часу: оновлення заявок і відображення змін без необхідності перезавантаження сторінки;
- TypeScript дозволяє уникнути помилок типізації на етапі розробки;

- Tailwind CSS пришвидшує верстку, забезпечуючи адаптивність і уніфікований стиль для всіх елементів.

Інтерфейс, створений за допомогою цього стеку, є зручним, швидким у взаємодії та візуально привабливим. Tailwind також сприяє дотриманню принципів дизайну без потреби в написанні додаткових CSS-класів.[16-21]

База даних: PostgreSQL

- PostgreSQL — одна з найпотужніших систем управління базами даних з відкритим кодом, яка підтримує ACID-транзакції, повнотекстовий пошук, збережені процедури та розширення. Вона була вибрана завдяки:
- Сумісності з Django: ORM у Django повністю підтримує PostgreSQL і дозволяє будувати запити високої складності;
- Надійності: системи резервного копіювання, журналювання транзакцій;
- Масштабованості: можливість горизонтального та вертикального масштабування;
- Безпеці: підтримка шифрування даних, доступу за ролями, перевірки автентичності.

СУБД виконує ключову роль у збереженні всієї інформації, пов’язаної з продажами, моделями автомобілів, користувачами та логами системи. PostgreSQL також дозволяє легко реалізувати аналітику на рівні БД.[22-25]

Таблиця 2.1 Порівняння з альтернативами

Компонент	Використано	Альтернатива	Коментар
Backend	Django	Node.js + Express	Django має вищу безпеку з коробки, вбудовану адмінпанель і ORM
Frontend	React + TypeScript	Angular/Vue.js	React більш гнучкий і популярний, легший для масштабування
CSS	Tailwind CSS	Bootstrap	Tailwind забезпечує кращу кастомізацію дизайну без перевантаження DOM
БД	PostgreSQL	MySQL / MongoDB	PostgreSQL більш надійна та функціональна для реляційних структур

Комбінація Django, React + TypeScript + Tailwind CSS і PostgreSQL утворює потужну платформу для побудови сучасного вебдодатку. Обраний стек технологій забезпечив не тільки швидку реалізацію функціоналу, але й

гарантує стабільну роботу, легкість у підтримці, масштабованість, а також знижує ризики, пов’язані з безпекою. Всі інструменти є відкритими, мають добру документацію та підтримуються активними спільнотами, що забезпечує перспективу подальшого розвитку додатку з мінімальними витратами.[12-25]

2.4. Розробка UML-діаграми класів

Проєктування UML-діаграми класів є одним із ключових етапів створення програмного забезпечення, адже саме вона дозволяє формалізувати логіку системи, її компоненти та взаємозв’язки між ними у вигляді наочної структурної моделі. У розроблюваному вебзастосунку UML-діаграма класів охоплює моделі, серіалізатори та представлення API, що разом формують цілісну архітектуру інформаційної системи. Реалізація побудована на основі Django REST Framework, що, своєю чергою, орієнтований на принципи об'єктно-орієнтованого програмування. Завдяки таким концепціям як інкапсуляція, наслідування, поліморфізм і абстракція, система отримує властивості масштабованості, гнучкості, повторного використання коду та високого рівня підтримки.

У структурі модулів застосунку виділяється три логічні частини: моделі, серіалізатори та представлення (views). Клас Car є базовою моделлю, що описує характеристики автомобіля, зокрема марку, модель, рік випуску, ціну, посилання на зображення, а також текстовий опис. До автомобіля додається модель CarSpecs, яка містить технічні характеристики, такі як потужність двигуна, прискорення, максимальна швидкість, тип трансмісії, тип двигуна, крутний момент, тип приводу, вага автомобіля та тип пального. Зв’язок між цими класами реалізований як один до одного, що означає, що кожен автомобіль має унікальний набір технічних характеристик. Додаткові зображення автомобіля зберігаються в моделі CarGallery, яка має зв’язок із класом Car через зовнішній ключ, тобто реалізується відношення "багато до одного". Модель ContactMessage використовується для зберігання повідомлень від користувачів, що надсилаються через форму зворотного

зв’язку. Вона містить поля для імені відправника, його електронної пошти, тексту повідомлення та мітки часу, яка фіксує момент відправлення.

Для перетворення моделей у зручний для API формат використовується набір серіалізаторів. Кожна модель має свій відповідний серіалізатор: CarSpecsSerializer, CarGallerySerializer та ContactMessageSerializer. Головним серіалізатором є CarSerializer, який об’єднує дані з пов’язаних моделей, включаючи вкладений серіалізатор для технічних характеристик і метод для динамічного отримання зображень з галереї. У цьому серіалізаторі також реалізовані методи створення та оновлення об’єктів.

Обробкою запитів до API займається клас CarAPI, який реалізує методи для отримання списку всіх автомобілів, отримання детальної інформації про конкретний автомобіль за його ідентифікатором, а також для обробки повідомлень, що надсилаються користувачами. Цей клас взаємодіє як із серіалізаторами, так і з моделями, виконуючи роль контролера, який поєднує логіку запитів із представленням даних у форматі, зручному для клієнтської частини застосунку.

Таким чином, UML-діаграма класів відображає логічну архітектуру вебзастосунку, визначаючи ключові компоненти системи, їхні обов’язки та типи взаємозв’язків. Це дозволяє забезпечити зрозумілу структуру проєкту на етапі розробки та полегшує подальший супровід і розширення функціональності. [26,27]

На рисунку 2.1 подано візуальне представлення цієї діаграми, яке ілюструє всі зв’язки між моделями, серіалізаторами та API в межах розроблюваної системи.

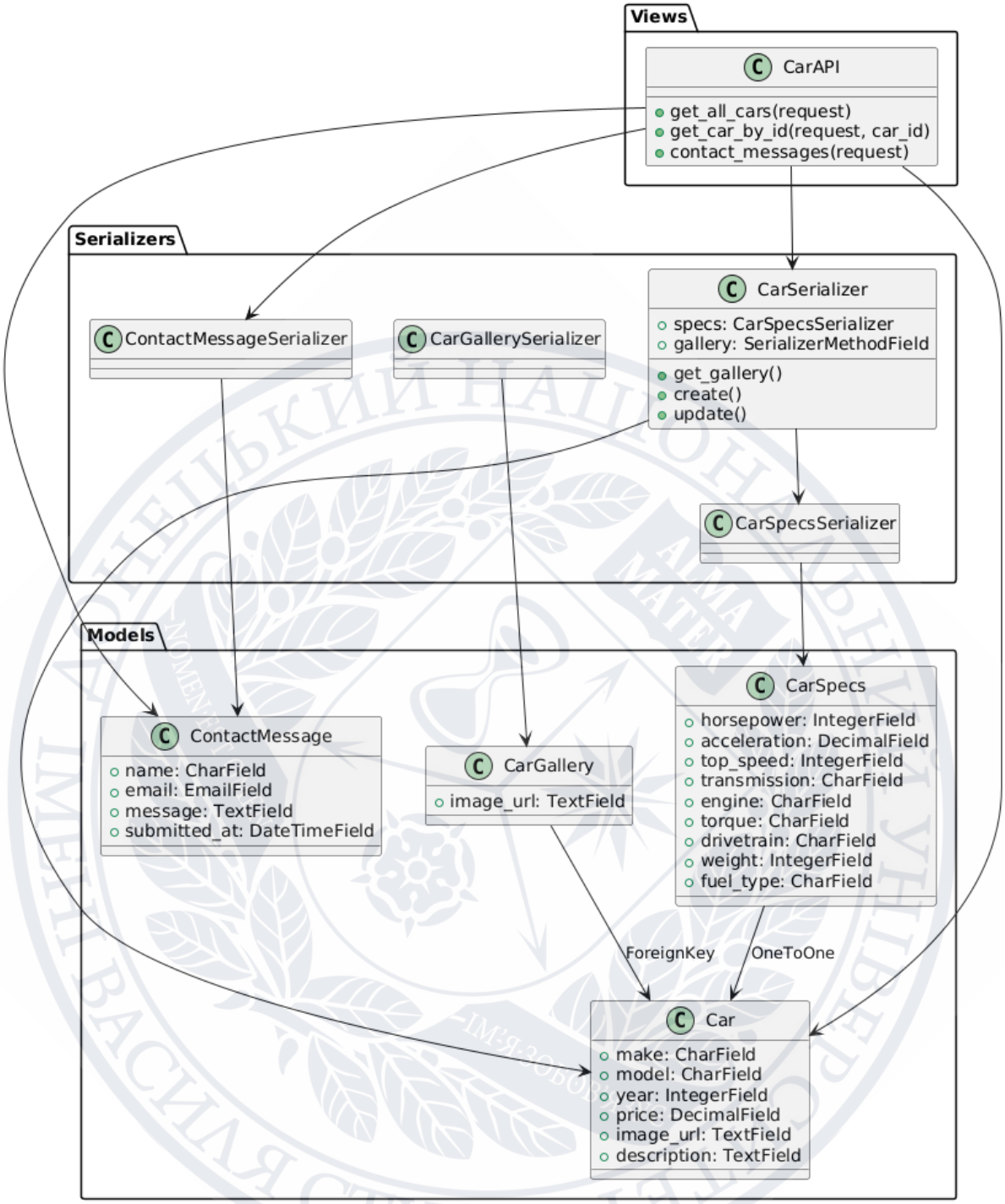

Рисунок 2.1 UML-діаграма класів вебзастосунку

2.5. Розробка ER-моделі бази даних

Розробка ER-моделі бази даних є важливим етапом проєктування системи, оскільки вона дозволяє наочно відобразити структуру бази даних, визначити основні сутності та зв'язки між ними. Для даного вебзастосунку була розроблена ER-модель, яка включає основні сутності, що описують автомобілі, їх технічні характеристики, зображення та повідомлення

користувачів. Кожна сутність відповідає за зберігання певних даних у базі, а зв'язки між ними забезпечують правильну організацію інформації та можливість ефективного взаємодії.

Основною сутністю є таблиця "Cars", яка зберігає дані про автомобілі. Вона містить ідентифікатор автомобіля (id), марку (make), модель (model), рік випуску (year), ціну (price), посилання на зображення автомобіля (image_url) та опис (description). Ці атрибути дозволяють детально описати основні характеристики автомобіля.

Таблиця "Car Specifications" зберігає технічні характеристики кожного автомобіля, таких як потужність (horsepower), прискорення (acceleration), максимальна швидкість (top_speed), тип трансмісії (transmission), тип двигуна (engine), крутний момент (torque), тип приводу (drivetrain), вага (weight) та тип пального (fuel_type). Зв'язок між таблицею "Cars" та таблицею "Car Specifications" реалізований через зовнішній ключ car_id, який посилається на ідентифікатор автомобіля у таблиці "Cars". Це дозволяє кожному автомобілю мати унікальний набір технічних характеристик.

Для зберігання зображень автомобілів передбачена таблиця "Car Gallery", яка містить посилання на додаткові зображення автомобіля (image_url). Кожен запис у цій таблиці пов'язаний з конкретним автомобілем через зовнішній ключ car_id, що дозволяє зберігати кілька зображень для одного автомобіля.

Таблиця "Contact Messages" містить інформацію про повідомлення, надіслані користувачами через форму зворотного зв’язку. Вона зберігає такі атрибути, як ім’я відправника (name), електронна пошта (email), текст повідомлення (message) та дата та час надсилання (submitted_at). Також між таблицею "Contact Messages" та таблицею "Cars" існує зв'язок, що позначений як "sent_inquiry". Це вказує на те, що кожне повідомлення користувача може

бути прив'язано до конкретного автомобіля, за яким клієнт зробив запит чи залишив відгук.

Таким чином, ER-модель бази даних забезпечує логічну організацію всіх необхідних даних для функціонування вебзастосунку. Зв'язки між таблицями гарантують, що всі сутності пов'язані між собою, а інформація зберігається ефективно та без дублювання.[28-30]

На рисунку 2.2 подано візуальне представлення цієї моделі, яке ілюструє всі зв’язки між сутностями.

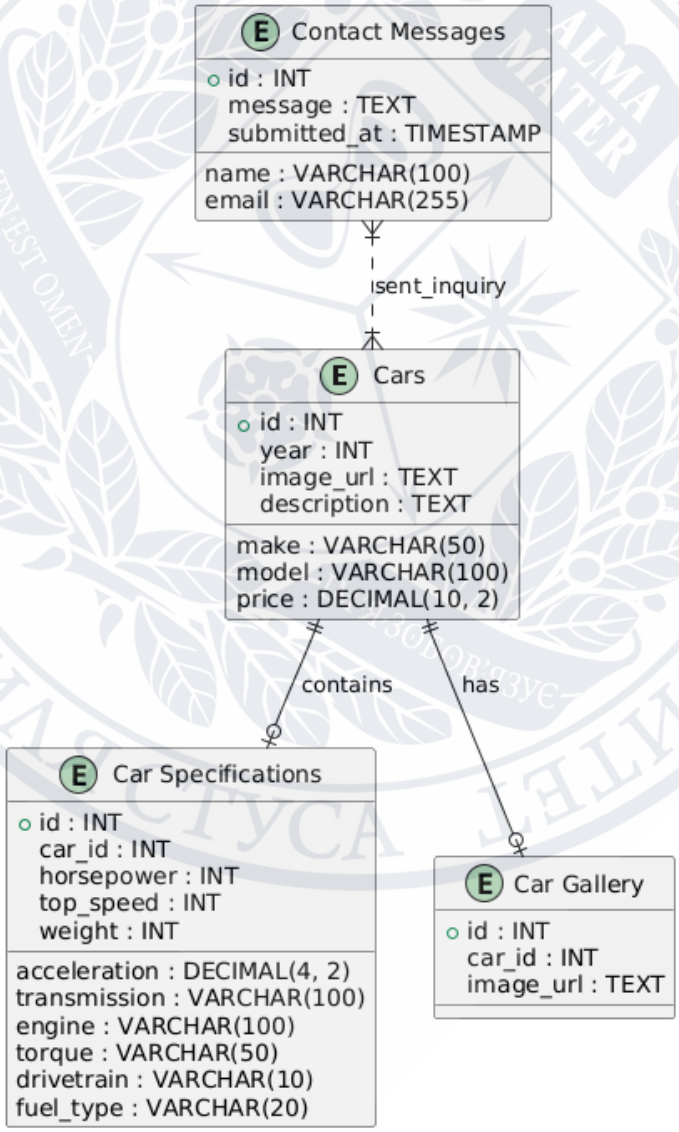

Рисунок 2.2 ER-модель бази даних вебзастосунку

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ ДЛЯ УПРАВЛІННЯ ДІЯЛЬНІСТЮ АВТОСАЛОНУ

3.1. Розробка бекенд частини вебдодатку

3.1.1 Інтеграція моделей вебзастосунку

Цей фрагмент коду відповідає за інтеграцію моделей вебзастосунку до адміністративної панелі Django, що дозволяє керувати даними через зручний графічний інтерфейс без потреби прямої роботи з базою даних.

У файлі admin.py імпортуються моделі Car, CarSpecs, CarGallery та ContactMessage з модуля models. Після цього кожна з моделей реєструється у адміністративній панелі за допомогою методу admin.site.register(...). Це забезпечує автоматичне створення відповідних форм для перегляду, створення, редагування та видалення записів у таблицях бази даних без написання додаткового коду.

```
from django.contrib import admin
from .models import Car, CarSpecs, CarGallery, ContactMessage
# Register models so they appear in Django Admin
admin.site.register(Car)
admin.site.register(CarSpecs)
admin.site.register(CarGallery)
admin.site.register(ContactMessage)
```

Завдяки цій реєстрації адміністратор сайту отримує змогу в реальному часі управляти даними про автомобілі, їх технічні характеристики, галереї зображень та повідомлення, отримані через форму зворотного зв’язку. Цей підхід значно спрощує супровід проєкту, тестування та заповнення бази на етапі розробки.

3.1.2 Основні моделі структури бази даних

У цьому фрагменті коду реалізовано основні моделі, що формують структуру бази даних вебзастосунку. Всі вони створені з використанням засобів Django ORM, що дозволяє описувати таблиці та зв'язки між ними у вигляді об'єктно-орієнтованого коду.

Центральною є модель Car, яка зберігає базову інформацію про автомобілі, зокрема марку, модель, рік випуску, ціну, зображення та опис:

```
class Car(models.Model):
    make = models.CharField(max_length=50)
    model = models.CharField(max_length=100)
    year = models.IntegerField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image_url = models.TextField()
    description = models.TextField()
    def __str__(self):
        return f"{self.make} {self.model} ({self.year})"
    class Meta:
        db_table = 'cars'
```

Метод str повертає зручне текстове представлення автомобіля для виводу в інтерфейсі адміністратора. Вказівка назви таблиці через Meta.db_table гарантує контроль над структурою бази даних.

Доповненням до цієї моделі є CarSpecs, яка реалізує технічні характеристики кожного авто. Вона пов'язана з моделлю Car через зв'язок один до одного (OneToOneField), що означає, що кожне авто має лише один набір специфікацій:

```
class CarSpecs(models.Model):
    car = models.OneToOneField(Car, on_delete=models.CASCADE, related_name="specs")
    horsepower = models.IntegerField()
    acceleration = models.DecimalField(max_digits=4, decimal_places=2)
    top_speed = models.IntegerField()
    transmission = models.CharField(max_length=100)
    engine = models.CharField(max_length=100)
    torque = models.CharField(max_length=50)
    drivetrain = models.CharField(max_length=10)
    weight = models.IntegerField()
    fuel_type = models.CharField(max_length=20)
    def __str__(self):
      return f"{self.car.make} {self.car.model} Specs"
    class Meta:
      db_table = 'car_specs'
```

Ця модель надає детальну технічну інформацію, яка дозволяє користувачам ознайомитись із ключовими параметрами авто.

Для підтримки мультимедійного контенту реалізована модель CarGallery, яка зберігає додаткові зображення автомобілів. Вона використовує зовнішній ключ (ForeignKey) для зв’язку з моделлю Car, що дозволяє кожному автомобілю мати кілька зображень:

```
class CarGallery(models.Model):
    car = models.ForeignKey(Car, on_delete=models.CASCADE, related_name="gallery")
```

```
    image_url = models.TextField()
    def __str__(self):
        return f"Image for {self.car.make} {self.car.model}"
    class Meta:
        db_table = 'car_gallery'
```

Останньою є модель ContactMessage, яка зберігає повідомлення користувачів, що надходять через форму зворотного зв'язку. Вона включає ім'я, email, сам текст повідомлення та час його надсилання:

```
class ContactMessage(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    message = models.TextField()
    submitted_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"Message from {self.name} ({self.email})"
    class Meta:
        db_table = 'contact_messages'
```

Завдяки полю submitted_at, яке автоматично фіксує дату й час створення, адміністратор може відстежувати надходження запитів у хронологічному порядку.

Ці моделі формують основу логічної структури даних системи, забезпечують ефективне зберігання й обробку інформації про автомобілі, їх характеристики, галереї зображень і запити користувачів.

3.1.3 Складні об'єкти Django

У цьому фрагменті реалізовано серіалізатори — спеціальні класи, що перетворюють складні об'єкти Django, такі як моделі, у формат, придатний для передачі через API, зазвичай у вигляді JSON. Вони також дозволяють

виконувати зворотну операцію — створення або оновлення об'єктів на основі отриманих даних.

Кожна з моделей, що використовувалась у попередньому блоці, має відповідний серіалізатор. Наприклад, серіалізатор CarSpecsSerializer забезпечує повну серіалізацію моделі технічних характеристик авто:

```
class CarSpecsSerializer(serializers.ModelSerializer):
    class Meta:
        model = CarSpecs
        fields = '__all__'
```

Аналогічно, CarGallerySerializer відповідає за серіалізацію зображень автомобіля, однак включає лише поле image_url, оскільки цього достатньо для відображення галереї на стороні клієнта:

```
class CarGallerySerializer(serializers.ModelSerializer):
    class Meta:
        model = CarGallery
        fields = ['image_url']
```

Для обробки повідомлень від користувачів через форму зворотного зв’язку використовується серіалізатор ContactMessageSerializer, який охоплює всі поля моделі:

```
class ContactMessageSerializer(serializers.ModelSerializer):
    class Meta:
        model = ContactMessage
        fields = '__all__'
```

Найскладнішим є серіалізатор CarSerializer, що включає вкладений серіалізатор CarSpecsSerializer, а також додатково використовує SerializerMethodField для галереї зображень. Це дозволяє сформувати галерею у вигляді простого списку посилань на зображення, а не повноцінних об’єктів:

```
class CarSerializer(serializers.ModelSerializer):
    specs = CarSpecsSerializer()
    gallery = serializers.SerializerMethodField()
    class Meta:
        model = Car
        fields = '__all__'
    def get_gallery(self, obj):
        """Return a flat list of image URLs instead of a list of objects"""
        return [image.image_url for image in obj.gallery.all()]
```

Використання вкладених серіалізаторів та спеціальних методів серіалізації забезпечує гнучкість і зручність у представленні складених об'єктів, що складаються з кількох пов’язаних моделей. Це важливо для фронтенд-частини застосунку, яка потребує структурованих і легко оброблюваних API-відповідей.

3.1.4 Маршрутизація API-запитів

Цей фрагмент коду визначає маршрутизацію для API-запитів у вебзастосунку. У файлі urls.py за допомогою функції path з модуля django.urls описано перелік URL-шляхів, що пов’язуються з відповідними обробниками запитів (view-функціями). Це дозволяє клієнтам надсилати HTTP-запити за визначеними адресами та отримувати відповідні дані або надсилати інформацію на сервер.

Файл містить наступну структуру маршрутів:

```
urlpatterns = [
    path('cars/', get_all_cars, name='get_all_cars'),
    path('cars/<int:car_id>/', get_car_by_id, name='get_car_by_id'),
    path('contact/', contact_messages, name='contact_messages'),
]
```

Перший маршрут 'cars/' викликає функцію get_all_cars, яка відповідає за повернення списку всіх доступних автомобілів. Це може бути використано, наприклад, для відображення каталогу на головній сторінці магазину або в результатах пошуку.

Другий маршрут 'cars/<int:car_id>/' — це динамічний шлях, у якому <int:car_id> є змінною частиною URL. Він дозволяє отримати детальну інформацію про конкретний автомобіль за його ідентифікатором, викликаючи функцію get_car_by_id.

Третій маршрут 'contact/' спрямований на обробку повідомлень з форми зворотного зв’язку. Він викликає функцію contact_messages, яка приймає дані від користувача, зберігає їх у базі даних і, за потреби, може відправляти підтвердження або повідомлення адміністраторам.

Таким чином, за допомогою цього файла формується базова структура REST API для взаємодії з даними про автомобілі та обробки зворотного зв’язку з клієнтами.

3.1.5 Логіка обробки API-запитів

Цей блок коду реалізує логіку обробки API-запитів у вигляді представлень (view-функцій) у Django REST Framework. Він міститься у файлі views.py і відповідає за три основні функціональні частини бекенду: отримання всіх автомобілів, отримання одного автомобіля за ID та обробку повідомлень із форми зворотного зв’язку.

Перша функція get_all_cars опрацьовує HTTP-запити типу GET і повертає повний список автомобілів, наявних у базі даних. Вона отримує всі екземпляри моделі Car, серіалізує їх за допомогою CarSerializer, і повертає результат у форматі JSON:

```
@api_view(['GET'])
def get_all_cars(request):
    """Retrieve all cars"""
    cars = Car.objects.all()
    serializer = CarSerializer(cars, many=True)
    return Response(serializer.data, content_type="application/json")
```

Друга функція get_car_by_id також обробляє запити типу GET, але з параметром car_id, що дозволяє отримати детальну інформацію про конкретний автомобіль. У разі, якщо автомобіль з таким ID не знайдено, повертається повідомлення про помилку з відповідним HTTP-статусом:

```
@api_view(['GET'])
def get_car_by_id(request, car_id):
    """Retrieve a single car by its ID"""
    try:
        car = Car.objects.get(pk=car_id)
        serializer = CarSerializer(car)
        return Response(serializer.data, content_type="application/json")
    except Car.DoesNotExist:
        return Response({'error': 'Car not found'}, status=404)
```

Третя функція contact_messages має дві гілки — для обробки POST та GET запитів. Якщо запит POST, функція приймає дані з контактної форми, перевіряє їх на валідність через ContactMessageSerializer, і, у разі успіху, зберігає їх у базу даних:

```
@csrf_exempt
@api_view(['POST', 'GET'])
def contact_messages(request):
    """Handles both GET and POST requests for contact messages"""
    if request.method == 'POST':
        serializer = ContactMessageSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Якщо ж запит GET, функція повертає всі повідомлення, впорядковані за датою від найновішого до найстарішого:

```
    elif request.method == 'GET':
        messages = ContactMessage.objects.all().order_by('-submitted_at')
        serializer = ContactMessageSerializer(messages, many=True)
        return Response(serializer.data)
```

Таким чином, цей блок реалізує повноцінний REST API для взаємодії з даними про автомобілі та повідомленнями користувачів, забезпечуючи як читання, так і створення записів.

3.2. Створення фронтенд частини вебдодатку

3.2.1 Структура проєкту

Структура проєкту організована за модульним принципом, що спрощує розробку, тестування та подальшу підтримку коду. Основна логіка додатку розподілена між такими ключовими директоріями:

app/ — містить усі сторінки додатку, включаючи головну сторінку (page.tsx), сторінку каталогу (catalog/page.tsx), контактну форму (contact/page.tsx), а також динамічні маршрути для перегляду деталей автомобіля (cars/[id]/page.tsx) та замовлення (order/[id]/page.tsx).

components/ — включає React-компоненти для побудови інтерфейсу, такі як кнопки, форми, картки товарів тощо.

styles/ — містить CSS-модулі для стилізації компонентів, що забезпечує ізольованість стилів та уникнення конфліктів.

hooks/ та lib/ — відповідають за кастомну бізнес-логіку, допоміжні функції та інтеграції з API.

Для забезпечення глобальних стилів використовується файл globals.css, який підключається в кореневому layout.tsx. Останній визначає базовий шаблон для всіх сторінок, включаючи спільні елементи, такі як заголовок, підвал або навігаційне меню.

3.2.2 Реалізація сторінки детального перегляду автомобіля

Сторінка детального перегляду (CarDetail) реалізована як клієнтський компонент Next.js з використанням TypeScript. Основна логіка компонента включає три ключові функціональності:

1. Завантаження даних

```
useEffect(() => {
  const loadCar = async () => {
```

```
    if (params.id) {
      const carData = await fetchCarById(params.id as string)
      setCar(carData)
      setLoading(false)
    }
   }
   loadCar()
  }, [params.id])
```

Цей ефект виконує запит до API при зміні параметра маршруту, завантажуючи дані конкретного автомобіля. Стани loading та car дозволяють керувати відображенням контенту.

2. Управління галереєю зображень:

```
const [currentImageIndex, setCurrentImageIndex] = useState(0)
const [selectedThumbnail, setSelectedThumbnail] = useState(0)
const nextImage = () => {
 setCurrentImageIndex((prev) => (prev + 1) % car.gallery.length)
 setSelectedThumbnail((prev) => (prev + 1) % car.gallery.length)
}
```

Функції nextImage та prevImage реалізують циклічну навігацію між зображеннями, синхронізуючи головне зображення з мініатюрами.

3. Вертикальна навігація між секціями

```
useEffect(() => {
  const handleScroll = (e: WheelEvent) => {
    e.preventDefault()
    if (isScrolling) return
    if (e.deltaY > 0 && currentSection < totalSections - 1) {
      setIsScrolling(true)
      setCurrentSection((prev) => prev + 1)
```

```
      setTimeout(() => setIsScrolling(false), 700)
    }
  }
  window.addEventListener("wheel", handleScroll, { passive: false })
  return () => window.removeEventListener("wheel", handleScroll)
}, [currentSection, isScrolling])
```

Цей механізм перехоплює події прокрутки для реалізації плавних переходів між повноекранними секціями.

3.2.3 Інтерфейс користувача

Hero-секція містить головне зображення з ефектами навігації:

```
<div className="relative h-full w-full">
  <Image
    src={car.gallery[currentImageIndex]}
    alt={`${car.make} ${car.model}`}
    fill
    priority
    className="object-cover"
  />
  <button
    onClick={prevImage}
    className="absolute left-4 top-1/2 -translate-y-1/2 p-2 bg-black/50 rounded-full"
  >
    <ChevronLeft size={24} />
  </button>
</div>
```

Секція характеристик використовує адаптивну сітку:

```
<div className="grid md:grid-cols-2 lg:grid-cols-3 gap-8">
```

```
{Object.entries(car.specs).map(([category, specs]) => (
 <div key={category} className="bg-white/5 p-6 rounded-lg">
  <h3 className="text-xl font-semibold mb-2">{category}</h3>
  <ul className="space-y-2">
   {Object.entries(specs).map(([key, value]) => (
    <li key={key} className="flex justify-between">
     <span className="text-gray-400">{key}</span>
     <span>{value}</span>
    </li>
   ))}
  </ul>
 </div>
))}
</div>
```

Галерея реалізована з ефектами hover:

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
 {car.gallery.map((image, index) => (
  <button
   key={image}
   onClick={() => {
    selectImage(index)
    setCurrentSection(0)
   }}
   className="relative aspect-[4/3] rounded-lg overflow-hidden group"
  >
   <Image
    src={image}
    fill
```

```
            className="object-cover transition-transform duration-300 group-hover:scale-110"
          />
        </button>
      ))}
    </div>
```

Компонент використовує кілька важливих оптимізацій:

1. Попереднє завантаження зображень через priority для головного зображення
2. Ліниве завантаження для мініатюр галереї
3. ARIA-атрибути для кнопок навігації
4. Обмеження анімацій через isScrolling для запобігання конфліктам

Завантажувальний стан реалізований з CSS-анімацією:

```
{loading && (
 <div className="h-screen flex items-center justify-center">
  <div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-white"></div>
 </div>
)}
```

Ця реалізація забезпечує плавну роботу навігації, чітку структуру даних та привабливий візуальний дизайн, що відповідає сучасним стандартам користувацького досвіду.

3.2.5 Сторінка каталогу автомобілів

Сторінка каталогу автомобілів реалізована як сучасний повноекранний інтерфейс з плавними переходами. Основна логіка компоненту включає три ключові частини:

1.Завантаження даних

```
const [cars, setCars] = useState<TransformedCar[]>([]);
```

```
const [loading, setLoading] = useState(true);
useEffect(() => {
  const fetchData = async () => {
    try {
      const data = await fetchCars();
      setCars(data);
    } catch (error) {
      console.error('Помилка завантаження:', error);
    } finally {
      setLoading(false);
    }
  };
  fetchData();
}, []);
```

Цей блок коду відповідає за отримання списку автомобілів з API. При ініціалізації компоненту викликається хук useEffect, який виконує асинхронний запит до сервера через функцію fetchCars. Під час завантаження встановлюється стан loading у значення true, що дозволяє відображати індикатор завантаження. У разі успішного отримання даних, вони зберігаються у стані cars. Якщо виникає помилка, вона логується у консоль, а стан loading змінюється на false у блоку finally, що припиняє відображення індикатора.

2. Навігаційна система

```
const [currentIndex, setCurrentIndex] = useState(0);
useEffect(() => {
  const handleWheel = (e: WheelEvent) => {
    e.preventDefault();
    if (e.deltaY > 0) {
      setCurrentIndex(prev => Math.min(prev + 1, cars.length - 1));
```

```
    } else {
      setCurrentIndex(prev => Math.max(prev - 1, 0));
    }
  };
  window.addEventListener('wheel', handleWheel);
  return () => window.removeEventListener('wheel', handleWheel);
}, [cars.length]);
```

Навігаційна система реалізована через обробку події прокручування колеса миші. Хук useEffect додає обробник події wheel до об'єкта вікна. При виявленні прокручування вниз (e.deltaY > 0) збільшується значення currentIndex, а при прокручуванні вгору - зменшується. Для запобігання виходу за межі масиву використовуються функції Math.min та Math.max. Перед виконанням змін відбувається виклик e.preventDefault(), що блокує стандартну поведінку прокручування сторінки.

3. Відображення контенту

```
      return (
        <div className="relative h-screen overflow-hidden">
          {cars.map((car, index) => (
            <div
              key={car.id}
              className="absolute inset-0 transition-transform duration-500"
              style={{ transform: `translateY(${100 * (index - currentIndex)}%)` }}
            >
              <Image
                src={car.imageUrl}
                alt={`${car.make} ${car.model}`}
                fill
                className="object-cover"
              />
```

```
            <div className="absolute inset-0 bg-black/40 flex flex-col justify-center items-center text-white">
              <h2 className="text-4xl font-bold">{car.make} {car.model}</h2>
              <p className="text-xl mt-2">{car.year} • ${car.price.toLocaleString()}</p>
            </div>
          </div>
        ))}
      </div>
    );
```

Цей блок відповідає за візуалізацію слайдера автомобілів. Кожен автомобіль відображається у повноекранному контейнері з абсолютним позиціонуванням. Трансформація translateY забезпечує вертикальний слайдинг, де значення зсуву розраховується на основі різниці між поточним індексом та індексом елемента. Для фону використовується компонент Image з Next.js, який оптимізує завантаження зображень. Поверх зображення додано напівпрозорий чорний оверлей з інформацією про автомобіль.

Для індикації поточного положення додано бічні маркери:

```
<div className="fixed right-8 top-1/2 transform -translate-y-1/2 flex flex-col gap-3">
  {cars.map((_, index) => (
    <button
      key={index}
      onClick={() => setCurrentIndex(index)}
      className={`w-3 h-3 rounded-full transition-all ${
        index === currentIndex ? 'bg-white scale-125' : 'bg-gray-400'
      }`}
    />
  ))}
</div>
```

Бічні маркери навігації реалізовані як вертикальний ряд кнопок, розташований фіксовано праворуч по центру екрану. Кожен маркер відповідає певному автомобілю в масиві cars. Активний маркер (що відповідає поточному індексу) виділяється білим кольором та збільшеним розміром. При кліку на маркер оновлюється стан currentIndex, що призводить до переходу до відповідного слайду. Анімація переходу забезпечується CSS-властивістю transition-all.

3.2.5 Реалізація контактної форми

Схема валідації форми визначена за допомогою бібліотеки Zod, що забезпечує типобезпеку та чіткі правила валідації:

```
const formSchema = z.object({
  name: z.string().min(2, {
    message: "Name must be at least 2 characters.",
  }),
  email: z.string().email({
    message: "Please enter a valid email address.",
  }),
  message: z.string().min(10, {
    message: "Message must be at least 10 characters.",
  }),
})
```

Ця схема вимагає, щоб ім'я містило мінімум 2 символи, email був у правильному форматі, а повідомлення - не менше 10 символів. Кожне правило включає повідомлення про помилку, яке буде відображатися користувачеві при невідповідності введених даних.

Управління станом форми реалізовано через хуки React Hook Form та Zod Resolver:

```
const form = useForm<z.infer<typeof formSchema>>({
```

```
  resolver: zodResolver(formSchema),
  defaultValues: {
    name: "",
    email: "",
    message: "",
  },
 })
```

Ця конфігурація інтегрує Zod схему валідації з React Hook Form, автоматично обробляючи перевірку даних при сабміті форми. Початкові значення полів встановлені як порожні рядки.

Обробка відправки форми включає кілька станів та обробку помилок:

```
 async function onSubmit(values: z.infer<typeof formSchema>) {
  setIsSubmitting(true)
  setError(null)
  try {
    const response = await fetch("/api/contact/", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(values),
    })
    if (!response.ok) {
      const errorData = await response.json()
      throw new Error(errorData.message || "Failed to submit the form.")
    }
    setIsSubmitted(true)
  } catch (err: any) {
    setError(err.message || "An unexpected error occurred.")
```

```
  } finally {
    setIsSubmitting(false)
  }
 }
```

Під час відправки активується стан isSubmitting, що блокує повторні сабміти. У разі успішної відправки встановлюється стан isSubmitted, який переключає відображення на повідомлення про успіх. При помилках відображається відповідне повідомлення.

Інтерфейс форми побудований з використанням компонентів UI бібліотеки:

```
<Form {...form}>
  <form  onSubmit={form.handleSubmit(onSubmit)}  className="space-y-8">
    <FormField
      control={form.control}
      name="name"
      render={({ field }) => (
        <FormItem>
          <FormLabel>Name</FormLabel>
          <FormControl>
            <Input placeholder="Your name" {...field} className="bg-white/10 text-white" />
          </FormControl>
          <FormMessage />
        </FormItem>
      )}
    />
    {/* Аналогічні поля для email та message */}
  </form>
</Form>
```

Кожне поле форми включає лейбл, інпут та місце для відображення повідомлень про помилки валідації. Стилізація виконана з урахуванням темної теми інтерфейсу. Повідомлення про успішну відправку показує просте повідомлення замість форми:

```
if (isSubmitted) {
  return (
    <div className="min-h-screen bg-black text-white flex items-center justify-center">
      <div className="text-center">
       <h1 className="text-4xl font-bold mb-4">Thank You!</h1>
       <p className="text-xl">We've received your message and will get back to you soon.</p>
      </div>
    </div>
  )
}
```

Цей блок рендериться при успішній відправці форми, інформуючи користувача про отримання повідомлення.

3.2.6 Сторінка оформлення замовлень

Схема валідації форми містить обов'язкові поля для оформлення замовлення:

```
const formSchema = z.object({
  name: z.string().min(2),
  email: z.string().email(),
  phone: z.string().min(10),
  address: z.string().min(10),
  comments: z.string().optional()
})
```

Схема валідації визначає правила для полів форми: ім'я (мінімум 2 символи), email (валідний формат), телефон (мінімум 10 цифр), адреса (мінімум 10 символів) та опціональні коментарі. Валідація виконується за допомогою бібліотеки Zod.

Завантаження даних про автомобіль відбувається при ініціалізації сторінки:

```
useEffect(() => {
  const loadCar = async () => {
    if (params.id) {
      const carData = await fetchCarById(params.id as string)
      setCar(carData)
    }
  }
  loadCar()
}, [params.id])
```

Завантаження даних про автомобіль відбувається через API при отриманні ID з параметрів URL. Хук useEffect викликає функцію fetchCarById при зміні params.id, результат зберігається у стані car.

Управління станом форми реалізовано через React Hook Form:

```
const form = useForm<z.infer<typeof formSchema>>({
  resolver: zodResolver(formSchema),
  defaultValues: {
    name: "",
    email: "",
    phone: "",
    address: "",
    comments: ""
  }
```

```
})
```

Управління формою реалізовано через react-hook-form з інтеграцією Zod для валідації. Форма містить початкові пусті значення полів та налаштування резолвера для обробки помилок.

Обробка відправки форми включає імітацію API-запиту:

```
function onSubmit(values: z.infer<typeof formSchema>) {
 setIsSubmitting(true)
 setTimeout(() => {
  console.log(values)
  setIsSubmitting(false)
  setIsSubmitted(true)
 }, 1000)
}
```

Обробка сабміту імітує відправку даних на сервер. Під час обробки активується стан isSubmitting, який блокує повторні відправки. Після успішного завершення встановлюється стан isSubmitted.

Інтерфейс форми відображає інформацію про автомобіль та поля для вводу:

```
<Form {...form}>
 <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6">
  <FormField
   control={form.control}
   name="name"
   render={({ field }) => (
    <FormItem>
     <FormLabel>Name</FormLabel>
     <FormControl>
```

```
            <Input placeholder="Your full name" {...field} />
          </FormControl>
        </FormItem>
      )}
    />
    {/* Інші поля форми */}
  </form>
</Form>
```

Інтерфейс форми включає поля для введення даних клієнта та відображення інформації про автомобіль. Кожне поле має лейбл, плейсхолдер та відображення помилок валідації.

Повідомлення про успішне замовлення показується після сабміту:

```
if (isSubmitted) {
  return (
    <div className="min-h-screen bg-black text-white flex items-center justify-center">
      <div className="text-center">
        <h1 className="text-4xl font-bold mb-4">Thank You for Your Order!</h1>
        <p className="text-xl">We've received your order...</p>
      </div>
    </div>
  )
}
```

Стан завантаження відображається поки дані про автомобіль не завантажені. Після успішного сабміту показується повідомлення про прийняття замовлення замість форми.

3.3. Розробка бази даних для вебдодатку

У цьому проєкті створена реляційна база даних, реалізована на основі СКБД PostgreSQL. Вона призначена для зберігання інформації про автомобілі, їх технічні характеристики, галереї зображень, а також повідомлення користувачів із контактної форми.

```
CREATE TABLE cars (
    id SERIAL PRIMARY KEY,
    make VARCHAR(50) NOT NULL,
    model VARCHAR(100) NOT NULL,
    year INT NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    image_url TEXT NOT NULL,
    description TEXT NOT NULL
);
```

Таблиця cars є основною в базі даних. Вона зберігає базову інформацію про автомобілі: виробника (make), модель (model), рік випуску (year), ціну (price), URL головного зображення (image_url) і текстовий опис (description). Поле id є первинним ключем і автоматично генерується завдяки типу SERIAL.

```
CREATE TABLE car_specs (
    id SERIAL PRIMARY KEY,
    car_id INT NOT NULL,
    horsepower INT NOT NULL,
    acceleration DECIMAL(4,2) NOT NULL,
    top_speed INT NOT NULL,
    transmission VARCHAR(100) NOT NULL,
    engine VARCHAR(100) NOT NULL,
    torque VARCHAR(50) NOT NULL,
    drivetrain VARCHAR(10) NOT NULL,
```

```
    weight INT NOT NULL,
    fuel_type VARCHAR(20) NOT NULL,
    FOREIGN KEY (car_id) REFERENCES cars(id) ON DELETE CASCADE
);
```

Таблиця car_specs містить технічні характеристики кожного автомобіля: потужність (horsepower), прискорення до 100 км/год (acceleration), максимальну швидкість (top_speed), тип трансмісії, двигуна, крутний момент, привід, масу та тип пального. Вона має зовнішній ключ car_id, який пов’язаний з таблицею cars, і забезпечує каскадне видалення (ON DELETE CASCADE) при видаленні запису з таблиці cars.

```
CREATE TABLE car_gallery (
    id SERIAL PRIMARY KEY,
    car_id INT NOT NULL,
    image_url TEXT NOT NULL,
    FOREIGN KEY (car_id) REFERENCES cars(id) ON DELETE CASCADE
);
```

Таблиця car_gallery дозволяє зберігати кілька додаткових зображень для кожного автомобіля. Поле image_url містить посилання на зображення, а поле car_id вказує на відповідну модель у таблиці cars. Завдяки зовнішньому ключу з опцією ON DELETE CASCADE, зображення буде автоматично видалено, якщо відповідний автомобіль видалити.

```
CREATE TABLE contact_messages (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    message TEXT NOT NULL,
```

```
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Таблиця contact_messages використовується для зберігання повідомлень, надісланих користувачами через форму зворотного зв'язку. Вона містить ім'я користувача (name), його email (email), текст повідомлення (message) та дату/час відправлення (submitted_at), яка автоматично встановлюється під час вставки.

У базі даних реалізовані такі зв'язки:

- car_specs.car_id - cars.id
- car_gallery.car_id - cars.id

Обидва зв'язки реалізовані через зовнішні ключі з правилом ON DELETE CASCADE, що дозволяє підтримувати цілісність даних при видаленні автомобіля з головної таблиці.

3.4. Тестування вебдодатку

У процесі розробки вебдодатку було проведено базове ручне тестування основних сторінок та функціональностей системи. Тестування здійснювалося у локальному середовищі за допомогою браузера Google Chrome. Метою тестування було перевірити працездатність інтерфейсу користувача, правильне відображення даних із бази, а також коректність роботи форми зворотного зв'язку. Усі перевірки проводились із використанням тестових даних, завантажених у базу.

Першочергово було перевірено завантаження головної сторінки. Після запуску додатку головна сторінка відкривається без помилок, коректно відображаються картки автомобілів із назвами, цінами та зображеннями, що завантажуються з бази даних. Усі візуальні елементи розташовані відповідно до дизайну, а зображення адаптуються до розміру екрану.

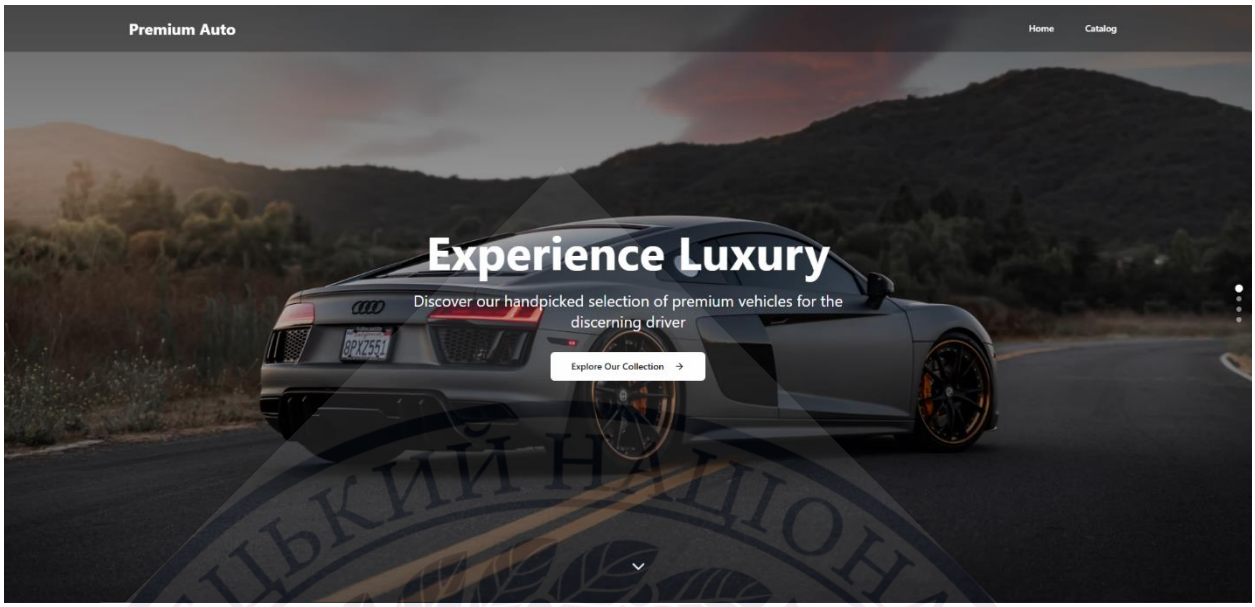

Рисунок 3.1 Головна сторінка вебдодатку

Далі було протестовано сторінку деталей автомобіля. При натисканні на картку певного авто відкривається індивідуальна сторінка з розширеною інформацією: технічні характеристики, детальний опис. Перевірено, що всі дані, включаючи динамічну галерею, завантажуються з відповідних таблиць бази даних.

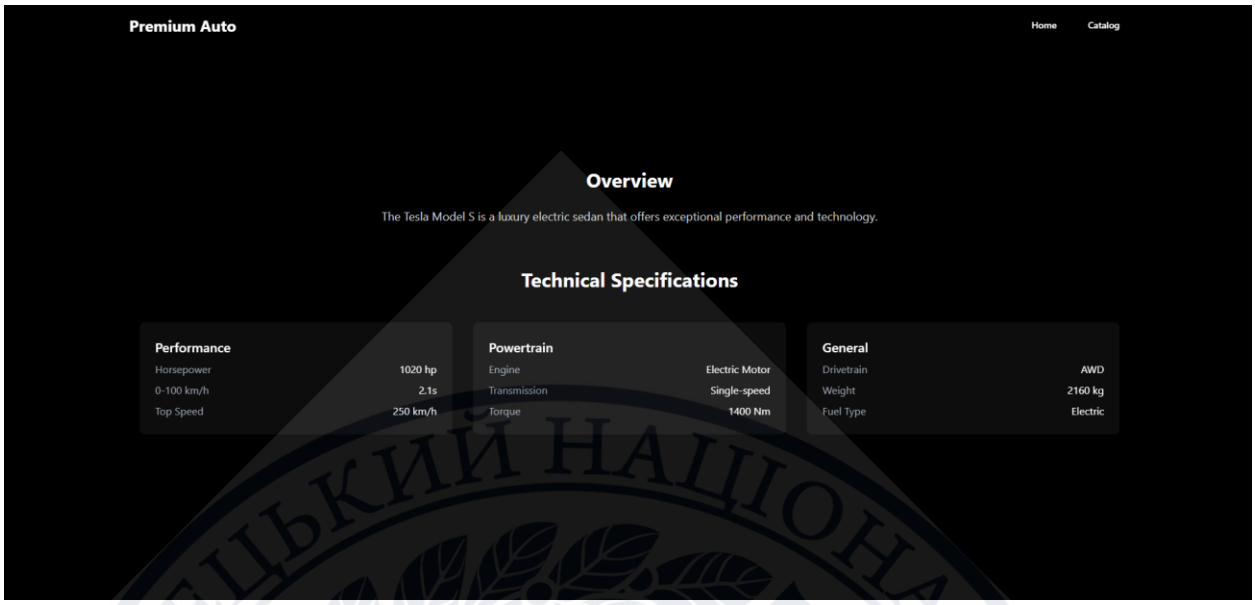

Рисунок 3.2 Сторінка деталей автомобіля

Також перевірена функціональність перегляду зображень у галереї. Користувач може переглядати кілька зображень одного автомобіля.

Зображення завантажуються з таблиці car_gallery відповідно до ID автомобіля, і правильно відображаються у вигляді слайдера або галереї.

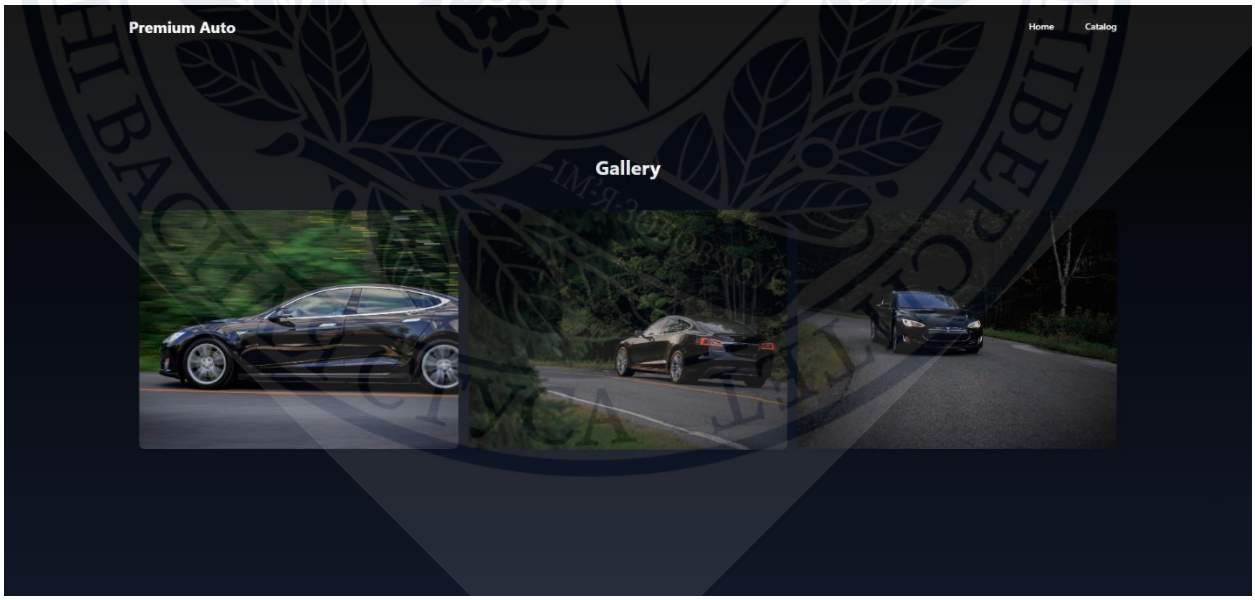

Рисунок 3.3 Сторінка зображень автомобіля

Окрему увагу приділено формі зворотного зв’язку. У формі перевірено, що всі поля є обов’язковими для заповнення. При введенні коректних даних та натисканні кнопки «Надіслати», повідомлення успішно зберігається у таблиці contact_messages, про що користувач отримує підтвердження. У разі введення некоректного email або порожніх полів система відображає повідомлення про помилку.

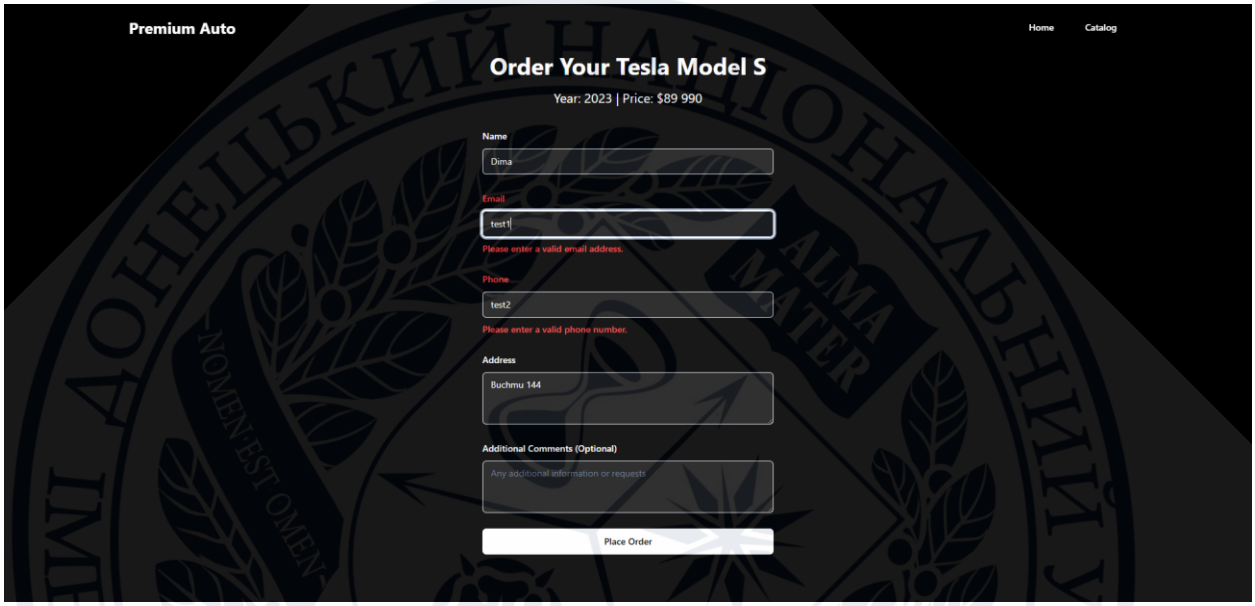

Рисунок 3.4 Введення помилкових даних

Крім того, було протестовано стабільність взаємодії з базою даних. Дані, які відображаються у додатку, повністю відповідають інформації, що зберігається у таблицях cars, car_specs та car_gallery. Видалення автомобіля із бази призводить до автоматичного видалення пов’язаних записів у таблицях специфікацій та зображень, що свідчить про правильну реалізацію зовнішніх ключів з каскадним видаленням.

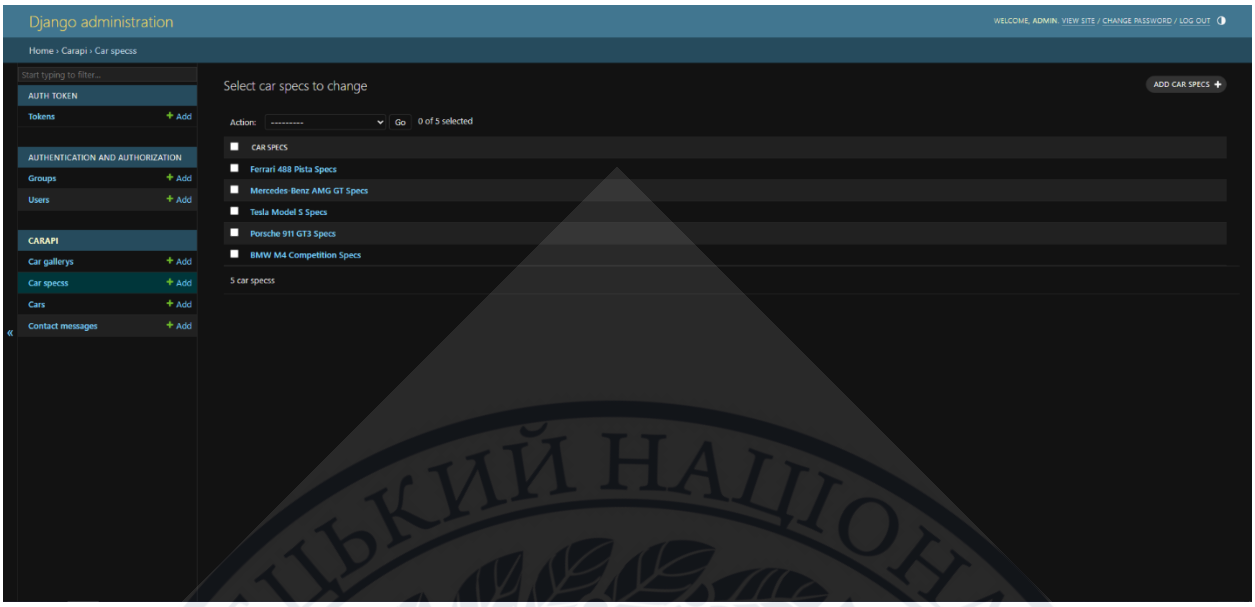

Рисунок 3.5 Сторінка адміністратора додатку

У результаті тестування підтверджено, що всі основні функції вебдодатку працюють стабільно, без критичних помилок. Інтерфейс відображається коректно на різних розмірах екранів, дані успішно інтегруються з базою, а форма зворотного зв’язку функціонує відповідно до очікувань. Додаток готовий до подальшого використання та розгортання на сервері.

ВИСНОВКИ

У ході ході виконання даної роботи було розроблено вебдодаток для управління діяльністю автосалону, який спрямований на автоматизацію ключових бізнес-процесів, підвищення ефективності роботи та забезпечення клієнтоорієнтованого підходу. Робота охопила три основні етапи: обґрунтування доцільності розробки, постановку задачі та вибір інструментів, а також програмну реалізацію вебдодатку.

На етапі обґрунтування доцільності (Розділ 1) було виявлено, що автосалони, які не використовують автоматизовані системи, стикаються з низкою проблем, таких як неефективне управління запасами, складність у веденні обліку клієнтів та відсутність оперативного доступу до даних. Використання вебдодатку дозволяє усунути ці недоліки завдяки централізації даних, зручному інтерфейсу та можливості швидкого аналізу інформації. Очікуваний вплив впровадження включає підвищення продуктивності працівників, зменшення часу на виконання рутинних операцій та покращення якості обслуговування клієнтів.

У Розділі 2 було сформульовано основні задачі розробки, серед яких: створення зручного інтерфейсу для управління автомобілями та замовленнями, а також забезпечення безпеки даних. Для вирішення цих задач було обрано сучасні інструменти, такі як Django для бекенду, React для фронтенду та PostgreSQL для бази даних. Розробка UML-діаграми класів та ER-моделі бази даних дозволила чітко визначити структуру системи та взаємозв’язки між її компонентами.

Програмна реалізація (Розділ 3) включала розробку бекенд та фронтенд частин вебдодатку. На етапі створення бекенду було реалізовано основні моделі бази даних, маршрутизацію API-запитів та логіку їх обробки. Фронтенд частина забезпечила зручний інтерфейс для користувачів, включаючи сторінки каталогу автомобілів, детального перегляду, оформлення замовлень

та контактну форму. Тестування вебдодатку підтвердило його функціональність, стабільність роботи та відповідність вимогам користувачів.

У результаті виконаної роботи було створено сучасний вебдодаток, який значно спрощує управління діяльністю автосалону. Він дозволяє автоматизувати рутинні процеси, забезпечує швидкий доступ до інформації та покращує взаємодію з клієнтами. Запровадження такого рішення може стати ключовим фактором у підвищенні конкурентоспроможності автосалону на ринку.

Майбутній розвиток проєкту може включати додавання нових функцій, таких інтеграція з зовнішніми сервісами (наприклад, платіжними системами або соціальними мережами), впровадження штучного інтелекту для аналізу даних та розширення можливостей аналітики. Крім того, важливим напрямом є оптимізація продуктивності системи та підвищення її безпеки для захисту конфіденційної інформації.

Таким чином, розроблений вебдодаток є ефективним інструментом для управління автосалоном, який відповідає сучасним вимогам та сприяє розвитку бізнесу.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Курдупов О.Л., Нескородєва Т.В. Дослідження систем рекомендацій при створенні веб-сайту автосалону. 2023.

2. Данилюк І.В., Бабала Л.В. Мобільні інформаційні технології у моделюванні управління послугами з технічного обслуговування та ремонту автомобілів. 2022, 122с.

3. Лук'янець Л.П. Розробка автоматизованої системи управління діяльністю автосалону. 2020, 64с.

4. Довгунь О.С. Автоматизація логістики: сучасні рішення та перспективи. 2017, 191с.

5. Чикусова М. Аналіз автомобільного ринку України. 2012, 38с.

6. Паньків, О. В. Мікросервісний вебзастосунок автосалону. 2023, 100с.

7. KeyCRM Blog(Оптимізація бізнес-процесів за допомогою автоматизації KeyCRM та ApiX-Drive.). URL: https://blog.keycrm.app/uk/optimizaciya-biznes-procesiv-za-dopomogoju-avtomatizacii/

8. CARIAD(Розробка клієнтського вебдодатку myVolkswagen). URL: https://cariad.technology/de/en/news/stories/agile-work-web-application.html

9. Spyne.ai(Топ-10 стратегій покращення взаємодії з клієнтами в автомобільній промисловості у 2025 році). URL: https://www.spyne.ai/blogs/automotive-customer-experience

10. Офіційний сайт Modera(Покращення взаємодії з клієнтами на автомобільних вебсайтах). URL: https://modera.com/automotive/enhance-customer-experience-on-automotive-websites/

11. Пею Р.К. Розробка інформаційного веб-сайту для автосалону. 2023, 108с.

12. Офіційна документація Django. URL: https://docs.djangoproject.com/en/5.2/intro/tutorial01/

13. Офіційний сайт W3Schools. URL: https://www.w3schools.com/django/

14. Офіційний сайт MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django

15. Офіційний сайт TutorialsPoint. URL: https://www.tutorialspoint.com/django/index.htm

16. Офіційний сайт CodersVibe. URL: https://www.codersvibe.com/react-app-with-typescript-and-tailwindcss-quick-setup-guide

17. Офіційний сайт GitHub. URL: https://github.com/Muhammad-Hasham-Khalid/react-typescript-tailwind-starter

18. Офіційний сайт Medium – Kaleab Dereje. URL: https://medium.com/@kaleabdereje/getting-started-with-react-typescript-and-tailwind-css-setting-up-your-development-environment-fa6761d87cef

19. Офіційний сайт FrontendShape. URL: https://frontendshape.com/post/install-setup-tailwind-css-react-18-typescript-vite

20.Офіційний сайт Larainfo. URL: https://larainfo.com/blogs/install-setup-vite-react-typescript-tailwind-css-3/

21.Офіційний сайт Medium – Uzeyr OZCAN. URL: https://medium.com/@muzeyr/building-a-react-typescript-application-with-vite-and-tailwind-css-94cafa2994d8

22. Офіційна документація PostgreSQL. URL: https://www.postgresql.org/docs/

23. PostgreSQL: Up and Running: A Practical Introduction to the Advanced Open Source Database. 2017, 312c.

24.Офіційний сайт W3Schools: PostgreSQL Tutorial. URL: https://www.w3schools.com/postgresql/

25.Офіційний сайт Neon.tech. URL: https://neon.tech/postgresql/tutorial

26. Фаулер М. UML 2.0 для розробників. 2003, 340с

27. Ларман Крег. Застосування UML 2.0 і шаблонів проектування. 2019, 736с.

28. Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H. V. Database Modeling and Design, 5th Edition. 2011, 352с.

29. Sikha Bagui, Richard Earp – Database Design Using Entity-Relationship Diagrams (3-тє видання). 2022, 388с.

30. Michael J. Hernandez. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. 2003, 611с.

ДОДАТКИ

Додаток А

Лістинг коду бекенду

```
from django.contrib import admin
from .models import Car, CarSpecs, CarGallery, ContactMessage
# Register models so they appear in Django Admin
admin.site.register(Car)
admin.site.register(CarSpecs)
admin.site.register(CarGallery)
admin.site.register(ContactMessage)

from django.db import models
class Car(models.Model):
    make = models.CharField(max_length=50)
    model = models.CharField(max_length=100)
    year = models.IntegerField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image_url = models.TextField()
    description = models.TextField()
    def __str__(self):
        return f"{self.make} {self.model} ({self.year})"
    class Meta:
        db_table = 'cars'
class CarSpecs(models.Model):
    car = models.OneToOneField(Car, on_delete=models.CASCADE, related_name="specs")
    horsepower = models.IntegerField()
    acceleration = models.DecimalField(max_digits=4, decimal_places=2)
    top_speed = models.IntegerField()
    transmission = models.CharField(max_length=100)
    engine = models.CharField(max_length=100)
    torque = models.CharField(max_length=50)
```

```
    drivetrain = models.CharField(max_length=10)
    weight = models.IntegerField()
    fuel_type = models.CharField(max_length=20)
    def __str__(self):
        return f"{self.car.make} {self.car.model} Specs"
    class Meta:
        db_table = 'car_specs'
class CarGallery(models.Model):
    car = models.ForeignKey(Car, on_delete=models.CASCADE, related_name="gallery")
    image_url = models.TextField()
    def __str__(self):
        return f"Image for {self.car.make} {self.car.model}"
    class Meta:
        db_table = 'car_gallery'
from django.db import models
class ContactMessage(models.Model):
    name = models.CharField(max_length=100)  # Ім'я користувача
    email = models.EmailField()              # Email користувача
    message = models.TextField()             # Повідомлення
    submitted_at = models.DateTimeField(auto_now_add=True)  # Дата/час створення
    def __str__(self):
        return f"Message from {self.name} ({self.email})"
    class Meta:
        db_table = 'contact_messages'  # Назва таблиці в базі даних


from rest_framework import serializers
from .models import Car, CarSpecs, CarGallery, ContactMessage=
class CarSpecsSerializer(serializers.ModelSerializer):
    class Meta:
        model = CarSpecs
        fields = '__all__'
```

```
class CarGallerySerializer(serializers.ModelSerializer):
    class Meta:
        model = CarGallery
        fields = ['image_url']
class ContactMessageSerializer(serializers.ModelSerializer):
    class Meta:
        model = ContactMessage
        fields = '__all__'
class CarSerializer(serializers.ModelSerializer):
    specs = CarSpecsSerializer()
    gallery = serializers.SerializerMethodField()
    class Meta:
        model = Car
        fields = '__all__'
    def get_gallery(self, obj):
        """Return a flat list of image URLs instead of a list of objects"""
        return [image.image_url for image in obj.gallery.all()]


from django.urls import path
from .views import get_all_cars, get_car_by_id, contact_messages
urlpatterns = [
    path('cars/', get_all_cars, name='get_all_cars'),
    path('cars/<int:car_id>/', get_car_by_id, name='get_car_by_id'),
    path('contact/', contact_messages, name='contact_messages'),
]
from rest_framework.response import Response
from rest_framework.decorators import api_view
from django.views.decorators.csrf import csrf_exempt
from rest_framework import status
from .models import Car, ContactMessage
from .serializers import CarSerializer, ContactMessageSerializer
```

```
@api_view(['GET'])
def get_all_cars(request):
    """Retrieve all cars"""
    cars = Car.objects.all()
    serializer = CarSerializer(cars, many=True)
    return Response(serializer.data, content_type="application/json")
@api_view(['GET'])
def get_car_by_id(request, car_id):
    """Retrieve a single car by its ID"""
    try:
        car = Car.objects.get(pk=car_id)
        serializer = CarSerializer(car)
        return Response(serializer.data, content_type="application/json")
    except Car.DoesNotExist:
        return Response({'error': 'Car not found'}, status=404)
@csrf_exempt
@api_view(['POST', 'GET'])
def contact_messages(request):
    """Handle contact messages"""
    """Handles both GET and POST requests for contact messages"""
    if request.method == 'POST':
        serializer = ContactMessageSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    elif request.method == 'GET':
        messages = ContactMessage.objects.all().order_by('-submitted_at')
        serializer = ContactMessageSerializer(messages, many=True)
        return Response(serializer.data)
```